

Perl Quick Reference

Variables, regex, file I/O, references, modules essentials

Basics

Hello World

```
#!/usr/bin/perl
use strict;
use warnings;
print "Hello, World!\n";
say "Hello, World!"; # with use feature 'say';
```

Run Perl

```
perl script.pl # run a file
perl -e 'print "hi\n"' # run inline
perl -ne 'print' file # process file line by line
```

Comments & Documentation

```
# single-line comment
=pod
Multi-line POD documentation
=cut
```

Variables

Sigils

\$scalar	Single value (string, number, reference)
@array	Ordered list of scalars
%hash	Key-value pairs
\$array[0]	Access single array element (scalar context)
%hash{key}	Access single hash value (scalar context)

Scalar Variables

```
my $name = "Perl"; # string
my $version = 5.40; # number
my $count = 42; # integer
my $undef; # undefined (undef)
my $combined = "$name v$version"; # interpolation
```

Context

```
my @arr = (1, 2, 3);
my $count = @arr; # scalar context: 3
my @copy = @arr; # list context: (1, 2, 3)
my $len = scalar @arr; # force scalar context
```

Special Variables

\$_	Default variable (topic)
@_	Subroutine arguments
!	System error message
\$@	Error from eval
\$0	Program name
@ARGV	Command-line arguments
%ENV	Environment variables

Operators

Comparison Operators

==, !=, <, >, <=, >=	Numeric comparison
eq, ne, lt, gt, le, ge	String comparison
<=>	Numeric spaceship (returns -1, 0, 1)
cmp	String spaceship
=~	Regex match / bind
!~	Regex negated match

String Operators

```
my $full = "Hello . " . "World"; # concatenation
my $line = "-" x 40; # repetition
my $len = length($full); # 11
```

Logical Operators

&& / and	Logical AND (low precedence: and)
 / or	Logical OR (low precedence: or)
//	Defined-or (returns left if defined)
! / not	Logical NOT
? :	Ternary conditional

Control Flow

Conditionals

```
if ($x > 0) { print "positive\n"; }
elsif ($x == 0) { print "zero\n"; }
else { print "negative\n"; }
print "yes\n" if $condition; # postfix if
print "no\n" unless $condition; # postfix unless
```

Loops

```
for my $i (0..9) { print "$i\n"; }
foreach my $item (@array) { print "$item\n"; }
while ($line = <STDIN>) { chomp $line; }
until ($done) { last if check(); }
```

Loop Control

next	Skip to next iteration (like continue)
last	Exit loop (like break)
redo	Restart current iteration
next LABEL	Skip to next iteration of labeled loop
last LABEL	Exit labeled loop

Given / When

```
use feature 'switch';
given ($status) {
    when ("ok") { say "success"; }
    when ("error") { say "failed"; }
    default { say "unknown"; }
}
```

Subroutines

Basic Subroutine

```
sub greet {
    my ($name) = @_;
    return "Hello, $name!";
}
my $msg = greet("Alice");
```

Default & Named Parameters

```
sub connect {
    my (%opts) = @_;
    my $host = $opts{host} // "localhost";
    my $port = $opts{port} // 5432;
    return "$host:$port";
}
connect(host => "db.example.com", port => 3306);
```

Subroutine References

```
my $double = sub { return $_[0] * 2; };
print $double->(5); # 10
my @sorted = sort { $a <=> $b } @nums;
```

Prototypes & Signatures

```
use feature 'signatures';
sub add($a, $b) { return $a + $b; }
sub greet($name, $greeting = "Hello") {
    return "$greeting, $name!";
}
```

Regex

Matching

```
if ($str =~ /pattern/) { print "matched\n"; }
if ($str =~ /(d+)/) { print "number: $1\n"; }
my @matches = ($str =~ /(w+)/g); # all matches
```

Substitution

```
$str =~ s/old/new/; # first occurrence
$str =~ s/old/new/g; # global (all occurrences)
$str =~ s/^\s+|\s+$//g; # trim whitespace
(my $clean = $str) =~ s/\W//g; # non-destructive copy
```

Modifiers

/i	Case-insensitive
/g	Global (all matches)
/m	Multi-line (^ and \$ match line boundaries)
/s	Single-line (. matches newline)
/x	Extended (allow whitespace and comments)

Common Patterns

\d, \D	Digit / non-digit
\w, \W	Word char / non-word char
\s, \S	Whitespace / non-whitespace
\b	Word boundary
(? ...)	Non-capturing group
(?<name> ...)	Named capture (access via \${name})

File I/O

Open & Read

```
open(my $fh, '<', 'data.txt') or die "Cannot open: $!";
while (my $line = <$fh>) {
    chomp $line;
    print "$line\n";
}
close($fh);
```

Write & Append

```
open(my $fh, '>', 'out.txt') or die "Cannot open: $!";
print $fh "Hello\n";
close($fh);
open(my $fh, '>>', 'log.txt') or die "Cannot open: $!";
print $fh "entry\n";
close($fh);
```

Slurp Entire File

```
use File::Slurp;
my $content = read_file('data.txt');
my @lines = read_file('data.txt', chomp => 1);
```

File Tests

-e \$path	File exists
-f \$path	Is a regular file
-d \$path	Is a directory
-r / -w / -x	Readable / writable / executable
-s \$path	File size in bytes (0 if empty)
-z \$path	File is zero size

Perl Quick Reference

Arrays & Hashes

Arrays

```
my @arr = (1, 2, 3, 4, 5);
push @arr, 6;           # append
my $last = pop @arr;   # remove last
my $first = shift @arr; # remove first
unshift @arr, 0;       # prepend
my @slice = @arr[1..3]; # slice
```

Array Functions

scalar @arr	Number of elements
push / pop	Add/remove from end
shift / unshift	Remove/add from beginning
splice(@a, 2, 1)	Remove 1 element at index 2
sort @arr	Sort alphabetically
reverse @arr	Reverse order
grep { /pat/ } @arr	Filter by pattern
map { \$_ * 2 } @arr	Transform each element
join(',', @arr)	Join into string

Hashes

```
my %user = (name => "Alice", age => 30);
$user{email} = "a@b.com"; # add pair
delete $user{age};        # remove pair
my @keys = keys %user;
my @vals = values %user;
```

Hash Iteration

```
while (my ($k, $v) = each %hash) {
    print "$k => $v\n";
}
for my $key (sort keys %hash) {
    print "$key: $hash{$key}\n";
}
```

References

Creating References

```
my $scalar_ref = \$name;
my $array_ref  = \@arr;
my $hash_ref   = \%hash;
my $anon_arr   = [1, 2, 3]; # anonymous array ref
my $anon_hash  = {a => 1, b => 2}; # anonymous hash ref
```

Dereferencing

```
print $$scalar_ref; # dereference scalar
print $array_ref->[0]; # arrow notation
print $hash_ref->{key}; # arrow notation
my @copy = @$array_ref; # dereference to array
my %copy = %$hash_ref; # dereference to hash
```

Complex Data Structures

```
my @users = (
    { name => "Alice", age => 30 },
    { name => "Bob",   age => 25 },
);
print $users[0]->{name}; # "Alice"
```

ref() Function

ref(\$r) eq 'SCALAR'	Reference to scalar
ref(\$r) eq 'ARRAY'	Reference to array
ref(\$r) eq 'HASH'	Reference to hash
ref(\$r) eq 'CODE'	Reference to subroutine

Modules

Using Modules

```
use strict;
use warnings;
use List::Util qw(sum max min);
use File::Basename;
use Cwd qw(abs_path);
```

Creating a Module

```
# MyModule.pm
package MyModule;
use Exporter 'import';
our @EXPORT_OK = qw(helper);
sub helper { return "help"; }
1; # module must return true
```

Common Core Modules

List::Util	sum, max, min, reduce, any, all
File::Basename	basename, dirname, fileparse
File::Path	make_path, remove_tree
Getopt::Long	Command-line option parsing
JSON	encode_json, decode_json
LWP::Simple	get(\$url) — simple HTTP client
Data::Dumper	Debug dump of data structures
Carp	croak, confess — better error messages

CPAN

```
cpan install Module::Name # install from CPAN
cpanm Module::Name       # cpanminus (faster)
perldoc Module::Name     # read module docs
```